

HAN-P031

ハードウェア／ソフトウェア協調検証方法

発明の背景

1. 発明の分野

本発明は、半導体装置に搭載されるハードウェア及びソフトウェアを協調検証 (Co-verification) する方法に関する。

2. 関連技術の説明

近年、S o C を搭載した機器が広く普及してきている。S o C とは、System o n a Chip の略で、コンピュータの主要機能を一つのチップ (半導体装置) に詰め込む技術、あるいは、当該技術によりコンピュータの主要機能を搭載したチップをいう。

図1は、かかるS o C の上流設計フローを示す図である。同図に示されるように、システム・レベルの設計が完了した後、アーキテクチャ・レベルの設計に移行する。アーキテクチャ・レベルの設計では、C P U (Central Processing Unit)、O S (Operating System)、バス (Bus) 等の基本部品の選択、ハードウェア及びソフトウェアへの機能分割、並びにハードウェア設計及びソフトウェア設計が行われる。そして、アーキテクチャ・レベルの設計により得られた基本部品、ハードウェア部品及びソフトウェア部品に対し、それらの検証モデルに基づくハードウェア／ソフトウェア協調検証が実行される。

一般に、かかるハードウェア／ソフトウェア協調検証においては、C P U の検証モデルとして、下記特許文献1～3に開示されるように、命令レベルでシミュレーションを行うI S S (Instruction Set Simulator, 命令セット・シミュレータ) が使用されている。このI S S は、C ベース言語で記述され、市販のシミュレータとしてS y s t e m - G (ガイオ・テクノロジー社製) 等が知られている。なお、本明細書において、「C ベース言語」とは、A N S I - C / C ++ の各種拡張言語、S p e c C、S y s t e m C のいずれかの言語を意味する。

又、C P U 専用メモリ、バス等のハードウェア部品についても、C ベース言語で記述された検証モデルが使用される。更に、その他のハードウェア部品につい

ても、Cベース言語を用いたビヘイビア (Behavior) 記述又はRTL (Register Transfer Level) 記述による検証モデルが使用される。なお、ビヘイビア記述とは、回路のあるまゝを記述したものであり、一方、RTL記述とは、レジスタの値が遷移していく様子を記述したものである。

一方、OS、ミドルウェア、割込みハンドラ、デバイス・ドライバ、タスク等のソフトウェア部品の検証モデルは、実論理即ちターゲットCPUのバイナリ・コード (命令コード) 自身で構成される。なお、本明細書において、「ターゲットCPU」とは、検証対象のSoC等の半導体装置に搭載されているCPU (例えば、ARMプロセッサ) を意味する。

以上のような検証モデル (ハードウェア部品、ソフトウェア部品) にテストベンチ及びCベース・シミュレータを加えることで、図2に示される如き従来の協調検証システム (ソフトウェア構成) が構成される。テストベンチは、テスト・データの入力、テスト・データの出力と期待値との比較、等を実行し、Cベース・シミュレータは、シミュレーション全体の動作を制御する。又、ISSは、ハードウェア部品からのマスク可能割込みINT (maskable INTerrupt) やテストベンチからのマスク不能割込みNMI (NonMaskable Interrupt) を受付けつつ、動作する。

このISSの機能は、命令レベル・シミュレーション、メモリ・アクセス (LOAD/STORE命令)、I/Oアクセス、及び割込み処理である。命令レベル・シミュレーション機能では、ターゲットCPUのバイナリ・コード・レベルでのシミュレーションを実行する。又、メモリ・アクセス機能では、バスへのREAD/WRITEアクセスを実行する。なお、バスがメモリ・アクセスを制御する。I/Oアクセス機能では、LOAD/STORE命令 (一回あたりのデータ転送量が小さい) を用いた、バスへのREAD/WRITEアクセスを実行する。なお、バスがI/O (ハードウェア部品) アクセスを制御する。又、割込み処理機能では、割込み (INT_i (i=1, ..., n)、NMI) の受付け、割込みハンドラの起動制御、実行中の処理の中断、中断した処理の再開、等を実行する。

ISSを使用した協調検証に関する先行技術文献としては、下記特許文献1～3の他に、下記の特許文献1及び2が存在する。なお、下記の特許文献3は

、後述する説明において引用される“Basic Block”に関するものであり、非特許文献4～6は、後述する説明において引用される“Fixed I/O Behaviorモデル”に関するものであり、非特許文献7～9は、C言語ベースの設計及び検証の技術動向に関するものである。

(特許文献1) 特開2000-259445号公報

(特許文献2) 特開2001-256072号公報

(特許文献3) 特開2002 175344号公報

(非特許文献1) 若林一敏：C言語によるLSI設計—動作合成とHW/SW協調検証の実際、NE Embedded Symposium 2002.

(非特許文献2) 黒川、池上、大坪、浅尾、桐ヶ谷、二栖、高橋、川津、新田、笠、若林、友部、高橋、向山、竹中：C言語ベースの動作合成を利用したシステムLSI設計手法の効果分析と考察、電子情報通信学会第15回軽井沢ワークショップ、pp. 131-142、Apr. 2002.

(非特許文献3) T. Sherwood, E. Perelman and B. Calder, “Basic Block Distribution Analysis to Find Periodic Behavior and Simulation Points in Applications”, in International Conference on Parallel Architectures and Compilation Techniques, Sept. 2001.

(非特許文献4) D. W. Knapp, T. Ly, D. MacMillen and R. Miller, “Behavioral Synthesis Methodology for HDL-Based Specification and Validation”, Proc. Design Automation Conf., June 1995.

(非特許文献5) T. Ly, D. W. Knapp, R. Miller and D. MacMillen, “Scheduling using Behavioral Templates”, Proc. Design Automation Conf., June 1995.

(非特許文献6) D. W. Knapp, “Behavioral Synthesis. Digital System Design using the Synopsys Behavioral Compiler”, Prentice Hall PTR.

(非特許文献7) L. Gauthier, S. Yoo and A. A. Jerraya, “Automatic Generation and Targeting of Application Specific Operating Systems and Embedded Systems Software”, Proc. Design Automation and Test in Europe, Mar. 2001.

(非特許文献 8) D. Lyonnard, S. Yoo, A. Baghdadi and A. A. Jerraya, "Automatic Generation of Application-Specific Architectures for Heterogeneous Multiprocessor System-on-Chip"; Proc. Design Automation Conf., June 2001.

(非特許文献 9) S. Yoo, G. Nicolescu, L. Gauthier and A. A. Jerraya, "Automatic Generation of Fast Timed Simulation Models for Operating Systems in SoC", Proc. Design Automation and Test in Europe, Mar. 2002.

上述のような、従来技術に係る、ISSを使用したハードウェア/ソフトウェア協調検証方法では、シミュレーションが命令レベルで実行され、即ち一命令ごとにその命令内容を解釈しながら実行されるとともに、メモリ・アクセスが必要となるため、シミュレーション時間即ち検証時間が大きくなるという問題がある。

発明の概要

本発明は、上述した問題点に鑑みてなされたものであり、その目的は、タイミングについてのシミュレーション精度を低下させることなく、一命令ごとの解釈と実行を不要とするCベースのネイティブ・コード・シミュレーションを実現することにより、シミュレーションの高速化を実現したハードウェア/ソフトウェア協調検証方法を提供することにある。

なお、以下の説明において、「Untimedソフトウェア部品」とは、その全ての処理をANSI-Cで記述したソフトウェア部品、又は、その全ての処理を実行するバイナリ・コードのソフトウェア部品を意味する。一方、「Timedソフトウェア部品」とは、その全ての処理を複数の単位処理に分割し、各単位処理を記述したANSI-C記述の後に、当該単位処理の実行時間を記述したCベース言語記述（実行時間挿入文）を追加したソフトウェア部品、又は、その全ての処理を複数の単位処理に分割し、各単位処理を実行するバイナリ・コードの後に、当該単位処理の実行時間を記述したCベース言語記述（実行時間挿入文）同等機能のバイナリ・コードを追加したソフトウェア部品を意味する。又、「ホストCPU」とは、協調検証を実行するパーソナル・コンピュータ（PC）又はワーク・ス

ーション (WS) に搭載されている CPU (例えば、Pentium (登録商標) プロセッサ) を意味する。

上記目的を達成するために、本発明の第一の面 (Aspect) によれば、ホスト CPU を使用して、一つのターゲット CPU 及び一つの OS が少なくとも搭載される半導体装置のハードウェア及びソフトウェアを協調検証する方法であって、(a) 検証モデルとしての C ベース言語記述の Timed ソフトウェア部品を入力してコンパイルし、検証モデルとしての C ベース言語記述のハードウェア部品を入力してコンパイルし、並びに該コンパイルされた Timed ソフトウェア部品及び該コンパイルされたハードウェア部品をリンクするステップと、(b) テストベンチを入力してコンパイルするステップと、(c) ステップ (a) により処理された検証モデルとステップ (b) により処理されたテストベンチとをリンクするステップと、(d) ステップ (c) で生成された実行プログラムに基づいてシミュレーションを実行するステップと、(e) ステップ (d) によるシミュレーションの結果を出力するステップと、を具備するハードウェア/ソフトウェア協調検証方法が提供される。

又、本発明の第二の面によれば、ホスト CPU を使用して、一つのターゲット CPU 及び一つの OS が少なくとも搭載される半導体装置のハードウェア及びソフトウェアを協調検証する方法であって、(a) 検証モデルとしてのホスト CPU 用バイナリ・コードの Timed ソフトウェア部品を入力し、検証モデルとしての C ベース言語記述のハードウェア部品を入力してコンパイルし、並びに該入力された Timed ソフトウェア部品及び該コンパイルされたハードウェア部品をリンクするステップと、(b) テストベンチを入力してコンパイルするステップと、(c) ステップ (a) により処理された検証モデルとステップ (b) により処理されたテストベンチとをリンクするステップと、(d) ステップ (c) で生成された実行プログラムに基づいてシミュレーションを実行するステップと、(e) ステップ (d) によるシミュレーションの結果を出力するステップと、を具備するハードウェア/ソフトウェア協調検証方法が提供される。

又、本発明の第三の面によれば、ホスト CPU を使用して、一つのターゲット CPU 及び一つの OS が少なくとも搭載される半導体装置のハードウェア及びソフトウェアを協調検証する方法であって、(a) 検証モデルとしての C ベース言語

記述のTimedソフトウェア部品を入力してコンパイルし、検証モデルとしてのホストCPU用バイナリ・コードのTimedソフトウェア部品を入力し、検証モデルとしてのCベース言語記述のハードウェア部品を入力してコンパイルし、並びに該コンパイルされ又は該入力されたTimedソフトウェア部品及び該コンパイルされたハードウェア部品をリンクするステップと、(b) テストベンチを入力してコンパイルするステップと、(c) ステップ(a)により処理された検証モデルとステップ(b)により処理されたテストベンチとをリンクするステップと、(d) ステップ(c)で生成された実行プログラムに基づいてシミュレーションを実行するステップと、(e) ステップ(d)によるシミュレーションの結果を出力するステップと、を具備するハードウェア/ソフトウェア協調検証方法が提供される。

又、本発明の第四の面によれば、前記本発明の第一又は第三の面による方法において、ANSI-C記述のUntimedソフトウェア部品からCベース言語記述のTimedソフトウェア部品を予め作成すべく、更に、ANSI-C記述のUntimedソフトウェア部品を入力して、Basic Block を認識し、制御点を挿入するステップと、該制御点を挿入されたUntimedソフトウェア部品をコンパイルしてターゲットCPU用バイナリ・コードを生成するステップと、該生成されたターゲットCPU用バイナリ・コードについて制御点間の実行時間を算出するステップと、該算出された実行時間に応じて、該制御点を挿入されたUntimedソフトウェア部品の各制御点へ実行時間挿入文を挿入し、Cベース言語記述のTimedソフトウェア部品として出力するステップと、が具備される。

又、本発明の第五の面によれば、前記本発明の第二又は第三の面による方法において、ターゲットCPU用バイナリ・コードのUntimedソフトウェア部品からホストCPU用バイナリ・コードのTimedソフトウェア部品を予め作成すべく、更に、ターゲットCPU用バイナリ・コードのUntimedソフトウェア部品を入力して、ホストCPU用バイナリ・コードのソフトウェア部品へと変換するステップと、該ホストCPU用バイナリ・コードのソフトウェア部品について、Basic Block を認識し、制御点を挿入するステップと、該制御点を挿入されたソフトウェア部品について制御点間の実行時間を算出するステップと、該算出された実行時間に応じて、該制御点を挿入されたソフトウェア部品の各制御点へ実行時間挿

入文同等機能のバイナリ・コードを挿入し、ホストCPU用バイナリ・コードのTimedソフトウェア部品として出力するステップと、が具備される。

図面の簡単な説明

本発明の更なる特徴及び利益は、添付図面を参照する次の説明から明らかとなる。図面において、

図1は、SoCの上流設計フローを示す図である。

図2は、従来のハードウェア/ソフトウェア協調検証システムの構成（ソフトウェア構成）を示すブロック図である。

図3は、本発明に係るハードウェア/ソフトウェア協調検証方法を実施するためのハードウェア環境を例示するブロック図である。

図4は、ソフトウェア部品の検証モデルの作成方法について説明するための図である。

図5は、ANSI-C記述のUntimedソフトウェア部品からCベース言語記述のTimedソフトウェア部品への変換を実行するプログラムの処理手順を説明するフローチャートである。

図6は、ANSI-C記述のUntimedソフトウェア部品からCベース言語記述のTimedソフトウェア部品への変換処理の内容について説明するための図である。

図7は、Cベース言語記述のTimedソフトウェア部品に基づくCベース・シミュレーションについて説明するための図である。

図8は、ターゲットCPU用バイナリ・コードのUntimedソフトウェア部品からホストCPU用バイナリ・コードのTimedソフトウェア部品への変換を実行するプログラムの処理手順を説明するフローチャートである。

図9は、本実施形態に係るハードウェア/ソフトウェア協調検証システムの構成（ソフトウェア構成）を示すブロック図である。

図10は、本実施形態に係るハードウェア/ソフトウェア協調検証の手順を説明するためのフローチャートである。

図11は、部品間の通信方式について説明するための図である。

ドを解釈することで累積処理時間 $T1 = t1$ が求められる。同様に、ノード c に相当する命令コードの実行後には累積処理時間 $T2 = t1 + t2$ が求められる一方、ノード f に相当する命令コードの実行後には累積処理時間 $T2 = t1 + t3$ が求められる。

そして、ノード g に相当する命令コードの実行後には、プログラムがノード c、ノード d 及びノード e のルートを走行した場合には累積処理時間 $T3 = t1 + t2 + t4$ が求められる一方、プログラムがノード c 及びノード f のルートを走行した場合には累積処理時間 $T3 = t1 + t3 + t4$ が求められることとなる。

ところで、ANSI-C 記述の Untimed ソフトウェア部品から C ベース言語記述の Timed ソフトウェア部品を作成する図 5 のソフトウェア部品検証モデル作成方法では、Untimed ソフトウェア部品のソース・コード (ANSI-C) が必要となるが、大半の OS やミドルウェアでは、その入手が困難である。そこで、本実施形態においては、ターゲット CPU 用バイナリ・コードによる Untimed ソフトウェア部品からホスト CPU 用バイナリ・コードによる Timed ソフトウェア部品への変換が行われる。

図 8 は、ターゲット CPU 用バイナリ・コードの Untimed ソフトウェア部品からホスト CPU 用バイナリ・コードの Timed ソフトウェア部品への変換を実行するプログラムの処理手順を説明するフローチャートである。まず、ステップ 210 では、ターゲット CPU 用バイナリ・コードの Untimed ソフトウェア部品 60 を入力して、ホスト CPU 用バイナリ・コードのソフトウェア部品 62 へと変換する。この変換は、Dynamic (Transitive Technologies 社製ツール) 等を利用して実現することができる。

次いで、ステップ 220 では、ホスト CPU 用バイナリ・コードのソフトウェア部品 62 について、前述したステップ 110 (図 5) と同様に、Basic Block を認識することにより、制御点を挿入されたソフトウェア部品 64 を出力する。

次いで、ステップ 230 では、前述したステップ 130 (図 5) と同様に、制御点を挿入されたソフトウェア部品 64 について制御点間の実行時間を算出する。その算出は、

図12は、マルチCPUシステムの検証モデルを例示する図である。

好適な実施例の説明

以下、添付図面を参照して本発明の実施形態について説明する。

図3は、本発明に係るハードウェア/ソフトウェア協調検証方法を実施するためのハードウェア環境を説明するための図である。同図に例示されるように、本発明に係る協調検証は、中央処理装置（CPU）12及び主記憶装置（MS）14を有するコンピュータ本体10、ディスプレイ20、キーボード22、マウス24、ハードディスク装置等からなる外部記憶装置30を備える通常のパーソナル・コンピュータ（PC）又はワーク・ステーション（WS）上で実行可能である。

CPU12は、協調検証を実行するホストCPUとして動作するものであり、例えば、Pentium（登録商標）プロセッサである。以下に説明される協調検証のためのプログラムは、CPU12によって実行される。又、各種のブータ、ファイル等は、外部記憶装置30から主記憶装置（MS）14にロードされて処理される。

次に、ハードウェア/ソフトウェア協調検証の対象となるSOC（半導体装置）に搭載される各種部品のうち、ソフトウェア部品の検証モデルの作成方法について、図4を用いて説明する。同図に示される例では、ソフトウェア部品のうち、OS/ミドルウェア及びタスクは、ANSI-Cのみで論理設計されているが、割込みハンドラ及びデバイス・ドライバについては、それらの設計論理にANSI-Cによる記述に加えてアセンブリ言語による記述が含まれる。そのため、まず、そのアセンブリ言語による記述部分がANSI-Cによる記述へと変換される。この変換は、人手によりなされる。

更に、デバイス・ドライバについては、バスへの直接READ/WRITEアクセスを実行するように、人手により変換される。即ち、従来のISSでは、一回あたりのデータ転送量が小さいLOAD/STORE命令を用いてI/Oアクセスがなされていたため低速であったが、本実施形態においては、I/Oアクセスの高速化が図られる。

最後に、全てANSI-Cにより記述されたOS/ミドルウェア、割込みハンドラ、デバイス・ドライバ及びタスクについて、Untimedソフトウェア部品（ANSI-C記述）からTimedソフトウェア部品（Cベース言語記述）への自動的な変換が、次に説明されるプログラムにより、実施される。

図5は、ANSI-C記述のUntimedソフトウェア部品からCベース言語記述のTimedソフトウェア部品への変換を実行するプログラムの処理手順を説明するフローチャートである。又、図6は、その変換処理の内容について説明するための図である。

まず、図5のステップ110では、ANSI-C記述のUntimedソフトウェア部品50を入力して、Basic Blockを認識し、制御点を挿入することにより、制御点を挿入されたUntimedソフトウェア部品（ANSI-C記述）52を出力する。このBasic Blockは、プログラムがストレートに走行する部分を指すものであり、その詳細は、上記非特許文献3に説明されている。そして、その認識されたBasic Blockの前後に制御点が挿入される。

即ち、図6（A）に示されるUntimedソフトウェア部品（ANSI-C記述）に対しては、ノードa及びノードbが一つのBasic Blockと認識され、ノードc、ノードd及びノードeが一つのBasic Blockと認識され、ノードc及びノードfが一つのBasic Blockと認識され、ノードgが一つのBasic Blockと認識される。そのため、図6（B）に示されるように、ノードhとノードcとの間、ノードeとノードgとの間、ノードfとノードgとの間、及びノードgの後に、それぞれ制御点が挿入される。

次いで、ステップ120では、かかる制御点を挿入されたUntimedソフトウェア部品（ANSI-C記述）52をコンパイルすることにより、ターゲットCPU用のバイナリ・コード54を生成する。

次いで、ステップ130では、上述のコンパイルの結果、生成されたターゲットCPU用バイナリ・コード（命令コード）54に基づいて、制御点間の実行時間を算出する。その算出は、

$k \Sigma$ [各命令のサイクル数]

なる演算式に基づいて行われる。ここで、係数 k は、キャッシュ・メモリのミス

ヒットに起因するオーバヘッド係数であり、本実施形態においてはキャッシュ・メモリ・モデルを設けないことから、統計的処理を可能とするために導入されたものである。

最後のステップ140では、制御点を挿入されたUntimedソフトウェア部品52の各制御点へ、ステップ130で算出された実行時間に応じた実行時間挿入文を挿入し、Cベース言語記述のTimedソフトウェア部品56として出力する。

例えば、ステップ130において、図6(B)に示されるノードa及びノードbの実行時間が t_1 と算出され、ノードc、ノードd及びノードeの実行時間が t_2 と算出され、ノードc及びノードfの実行時間が t_3 と算出され、ノードgの実行時間が t_4 と算出されたとする。その場合には、図6(C)に示されるように、ノードbとノードcとの間の制御点には、実行時間挿入文として `waitfor(t_1)` が挿入される。同様に、ノードeとノードgとの間の制御点には実行時間挿入文として `waitfor(t_2)` が挿入され、ノードfとノードgとの間の制御点には実行時間挿入文として `waitfor(t_3)` が挿入され、ノードgの後の制御点には実行時間挿入文として `waitfor(t_4)` が挿入されることとなる。

図7は、このようにして作成されたCベース言語記述のTimedソフトウェア部品56に基づくCベース・シミュレーションについて説明するための図である。Cベース言語記述のTimedソフトウェア部品56をコンパイルしてホストCPU用バイナリ・コードを生成することにより、ホストCPU12(図3)におけるネイティブ・コード・シミュレーション(Native Code Simulation)が可能となる。

即ち、Cベース言語記述のTimedソフトウェア部品56をコンパイルして生成されたホストCPU用バイナリ・コードをホストCPUで実行させることにより、一命令ごとにその命令内容を解釈してシミュレーションを実行するISSの場合に比較して、100~1000倍程度の高速化を図ることができる。

一方、シミュレーションの過程において、`waitfor` 文が出現したところで、シミュレータがその内容を解釈することにより、命令実行時間の管理が可能となるため、タイミングについてのシミュレーション精度も維持することができる。即ち、ノードbに相当する命令コードの実行後には、`waitfor(t_1)` の命令コ

28. Jan. 2004 17:04

A. Aoki, Ishida & 81-3-5470-1911

NO. 1760 P. 15/35

$k \Sigma$ [各命令のサイクル数]

なる演算式に基づいて行われる。ここで、係数 k は、キャッシュ・メモリのミスヒットに起因するオーバヘッド係数である。

最後のステップ 240 では、制御点を挿入されたソフトウェア部品 64 の各制御点へ、算出された実行時間に応じた実行時間挿入と同等機能のバイナリ・コードを挿入し、ホスト CPU 用バイナリ・コードの Timed ソフトウェア部品 66 として出力する。

次に、ハードウェア/ソフトウェア協調検証の対象となる SoC (半導体装置) に搭載される各種部品のうち、ハードウェア部品の検証モデルの作成方法について説明する。前述のように、従来のハードウェア/ソフトウェア協調検証においては、CPU 及び CPU 専用メモリの検証モデルが準備されたが、本実施形態においては、それらの検証モデルは作成されず、その代わりに、ISS (Instruction Set Simulator) における割込み処理部を独立化させた IRS (Interrupt Routine Scheduler) が新たに導入される。この IRS は、C ベース言語で記述される。又、バスの検証モデルは、従来と同様に、C ベース言語で新規に作成される。

又、C ベース言語を用いて論理設計されたビヘイビア (Behavior) 記述のハードウェア部品は、動作合成ツールの拡張機能を利用して、C ベース言語記述から Fixed I/O Behavior モデルへと自動的に変換されることにより、その検証モデル (C ベース言語) が生成される。この Fixed I/O Behavior モデルは、Basic Block を利用した Timed ハードウェア・モデルと同等のものであり、その詳細は、上記の非特許文献 4～6 に説明されている。

又、Verilog/VHDL を用いて論理設計された RTL (Register Transfer Level) 記述のハードウェア部品は、IIDL Import (CoWare 社製ツール) 等を利用して、RTL 記述から RTL-C ベース言語モデルへと自動的に変換されることにより、その検証モデル (C ベース言語) が生成される。この RTL-C ベース言語モデルは、FSM (Finite State Machine, 有限状態機械) の 1 ステートが 1 クロックの動作を表現するものである。

以上のようにして作成されたソフトウェア部品及びハードウェア部品の検証モ

デルにテストベンチ及びCベース・シミュレータを加えることで、本実施形態に係る協調検証システム（ソフトウェア構成）が図9に示されるように構成される。なお、前述のように、テストベンチは、テスト・データの入力、テスト・データの出力と期待値との比較、等を実行し、Cベース・シミュレータは、シミュレーション全体の動作を制御する。IRSは、CPU割込み回路相当の機能を実現するものである。又INTは、マスク可能割込み（maskable INTerrupt）、NMIは、マスク不能割込み（NonMaskable Interrupt）である。

図10は、図3に示されるハードウェア環境及び図9に示されるソフトウェア構成の下で実行される、SOCに対するハードウェア/ソフトウェア協調検証の手順を説明するためのフローチャートである。まず、ステップ310では、検証モデルの入力、必要なコンパイル、及びリンクを実行する。

即ち、検証モデルとして、ホストCPU用バイナリ・コードのTimedソフトウェア部品、Cベース言語記述のTimedソフトウェア部品、Cベース言語記述のバス、Cベース言語記述の一般ハードウェア部品、及びCベース言語記述のIRSを入力する。なお、Timedソフトウェア部品としては、ホストCPU用バイナリ・コードのTimedソフトウェア部品のみが入力される場合、Cベース言語記述のTimedソフトウェア部品のみが入力される場合、及びそれらの双方が入力される場合がある。なお、ソフトウェア部品には、前述のように、OS/ミドルウェア、割込みハンドラ、デバイス・ドライバ、タスク等が含まれる。又、一般ハードウェア部品は、CPU、CPU専用メモリ及びバスを除くハードウェア部品である。

そして、Cベース言語記述のTimedソフトウェア部品、Cベース言語記述のバス、Cベース言語記述の一般ハードウェア部品、及びCベース言語記述のIRSについては、そのコンパイルを行う一方、ホストCPU用バイナリ・コードのTimedソフトウェア部品についてはコンパイルの必要はない。かかるコンパイルを実施した後、全てのソフトウェア部品及びハードウェア部品をリンクする。

次いで、ステップ320では、テストベンチを入力してコンパイルする。更に、ステップ330では、ステップ310において処理された各部品即ち検証モデルと、ステップ320において処理されたテストベンチとをリンクする。次いで

、ステップ340では、Cベース・シミュレータによる制御の下、ステップ330で生成された実行プログラムにより、図7で説明されたシミュレーションが実行される。最後のステップ350では、そのシミュレーションの結果がディスプレイ20等に出力されて、協調検証が終了する。

最後に、本実施形態における部品間の通信方式について、図11に基づき説明すると、ソフトウェア部品とソフトウェア部品との間の通信方式としては、ANSI-C通信方式(Semaphore, MailBox, EventFlag)が採用される。一方、ハードウェア部品とハードウェア部品との間の通信方式としては、SystemC通信方式(sc_event)が採用される。又、ソフトウェア部品とハードウェア部品との間の通信方式としては、ハードウェア起動時にあつてはSystemC通信方式(sc_event)が採用される。一方、ハードウェア動作終了時にあつては割り込み方式が採用される。

一方、非特許文献7～9に記載されているC言語ベースの設計及び検証技術の実施形態における部品間の通信方式について説明すると、ソフトウェア部品とソフトウェア部品との間の通信方式及びハードウェア部品とハードウェア部品との間の通信方式としては、SystemC通信方式(sc_event)が採用されている。又、ソフトウェア部品とハードウェア部品との間の通信方式としては、ハードウェア起動時にあつてはSystemC通信方式(sc_event)が採用されている。一方、ハードウェア動作終了時にあつてはポーリング方式が採用されている。

以上、本発明の実施形態について述べてきたが、もちろん本発明はこれに限定されるものではなく、例えば、図12に示されるような、マルチCPUシステムの検証モデルにも、本発明は適用可能である。

以上説明したように、本発明によれば、Cベースのネイティブ・コード・シミュレーションが実現されるため、従来のISSを使用した方法に比較して、シミュレーション性能(命令数/秒)が従来の $10^2 \sim 10^3$ 倍程度向上し、ハードウェア/ソフトウェア協調検証におけるシミュレーションの高速化が図られる。しかも、Timedソフトウェア部品に基づく時間管理が行われることで、タイミング検証の精度も維持される。したがって、本発明は、ハードウェア/ソフトウェア協調検証(特に、ソフトウェアの検証)における工数の低減に大きく寄与するも

のである。

本発明は、その精神及び本質的特徴から離れることなく他の特定の形態で具体化され得る。それ故、本実施例は、全ての点で例示的かつ非限定的なものとみなされるべきであり、本発明の範囲は、前述の説明によってではなく、添付の請求の範囲によって示され、それ故、当該請求の範囲の意義及び均等の範囲に帰着する全ての変更は、そこに包含されるものとなる。

請求の範囲

1. ホストCPUを使用して、一つのターゲットCPU及び一つのOSが少なくとも搭載される半導体装置のハードウェア及びソフトウェアを協調検証する方法であって、

(a) 検証モデルとしてのCベース言語記述のTimedソフトウェア部品を入力してコンパイルし、検証モデルとしてのCベース言語記述のハードウェア部品を入力してコンパイルし、並びに該コンパイルされたTimedソフトウェア部品及び該コンパイルされたハードウェア部品をリンクするステップと、

(b) テストベンチを入力してコンパイルするステップと、

(c) ステップ(a)により処理された検証モデルとステップ(b)により処理されたテストベンチとをリンクするステップと、

(d) ステップ(c)で生成された実行プログラムに基づいてシミュレーションを実行するステップと、

(e) ステップ(d)によるシミュレーションの結果を出力するステップと、

を具備するハードウェア/ソフトウェア協調検証方法。

2. ホストCPUを使用して、一つのターゲットCPU及び一つのOSが少なくとも搭載される半導体装置のハードウェア及びソフトウェアを協調検証する方法であって、

(a) 検証モデルとしてのホストCPU用バイナリ・コードのTimedソフトウェア部品を入力し、検証モデルとしてのCベース言語記述のハードウェア部品を入力してコンパイルし、並びに該入力されたTimedソフトウェア部品及び該コンパイルされたハードウェア部品をリンクするステップと、

(b) テストベンチを入力してコンパイルするステップと、

(c) ステップ(a)により処理された検証モデルとステップ(b)により処理されたテストベンチとをリンクするステップと、

(d) ステップ(c)で生成された実行プログラムに基づいてシミュレーションを実行するステップと、

(e) ステップ(d)によるシミュレーションの結果を出力するステップと、

を具備するハードウェア/ソフトウェア協調検証方法。

3. ホストCPUを使用して、一つのターゲットCPU及び一つのOSが少なくとも搭載される半導体装置のハードウェア及びソフトウェアを協調検証する方法であって、

(a) 検証モデルとしてのCベース言語記述のTimedソフトウェア部品を入力してコンパイルし、検証モデルとしてのホストCPU用バイナリ・コードのTimedソフトウェア部品を入力し、検証モデルとしてのCベース言語記述のハードウェア部品を入力してコンパイルし、並びに該コンパイルされ又は該入力されたTimedソフトウェア部品及び該コンパイルされたハードウェア部品をリンクするステップと、

(b) テストベンチを入力してコンパイルするステップと、

(c) ステップ(a)により処理された検証モデルとステップ(b)により処理されたテストベンチとをリンクするステップと、

(d) ステップ(c)で生成された実行プログラムに基づいてシミュレーションを実行するステップと、

(e) ステップ(d)によるシミュレーションの結果を出力するステップと、

を具備するハードウェア/ソフトウェア協調検証方法。

4. ANSI-C記述のUntimedソフトウェア部品からCベース言語記述のTimedソフトウェア部品を予め作成すべく、更に、

ANSI-C記述のUntimedソフトウェア部品を入力して、Basic Block を認識し、制御点を挿入するステップと、

該制御点を挿入されたUntimedソフトウェア部品をコンパイルしてターゲットCPU用バイナリ・コードを生成するステップと、

該生成されたターゲットCPU用バイナリ・コードについて制御点間の実行時間を算出するステップと、

該算出された実行時間に応じて、該制御点を挿入されたUntimedソフトウェア部品の各制御点へ実行時間挿入文を挿入し、Cベース言語記述のTimedソフトウェア部品として出力するステップと、

を具備する、請求項1又は請求項3に記載のハードウェア/ソフトウェア協調検証方法。

5. ターゲットCPU用バイナリ・コードのUntimedソフトウェア部品から
ホストCPU用バイナリ・コードのTimedソフトウェア部品を予め作成すべく、
更に、

ターゲットCPU用バイナリ・コードのUntimedソフトウェア部品を入力して
、ホストCPU用バイナリ・コードのソフトウェア部品へと変換するステップと

、
該ホストCPU用バイナリ・コードのソフトウェア部品について、Basic Block
を認識し、制御点を挿入するステップと、

該制御点を挿入されたソフトウェア部品について制御点間の実行時間を算出する
ステップと、

該算出された実行時間に応じて、該制御点を挿入されたソフトウェア部品の各
制御点へ実行時間挿入と同等機能のバイナリ・コードを挿入し、ホストCPU用
バイナリ・コードのTimedソフトウェア部品として出力するステップと、

を具備する、請求項2又は請求項3に記載のハードウェア/ソフトウェア協調
検証方法。

ハードウェア／ソフトウェア協調検証方法

開示の要約

タイミング検証の精度を低下させることなく、Cベースのネイティブ・コード・シミュレーションを実現することにより、シミュレーションの高速化を実現したハードウェア／ソフトウェア協調検証方法。この方法は、ホストCPUを使用して、一つのターゲットCPU及び一つのOSが少なくとも搭載される半導体装置のハードウェア及びソフトウェアを協調検証する方法であって、まず、検証モデルとして、Cベース言語記述又はホストCPU用バイナリ・コードのTimedソフトウェア部品及びCベース言語記述のハードウェア部品を入力し、必要なコンパイルを実行してそれらをリンクする。次いで、テストベンチを入力してコンパイルする。次いで、各部品とテストベンチとをリンクし、シミュレーションを実行して、その結果を出力する。